القوائم Lists

المكدس Stack :

التاريخ History :

المكدس stack قدم سنة 1955 ، وسجل كبراءة اختراع سنة 1957، من قبل الألماني فريدريك Friedrich L. Bauer .

التعريف Definition :

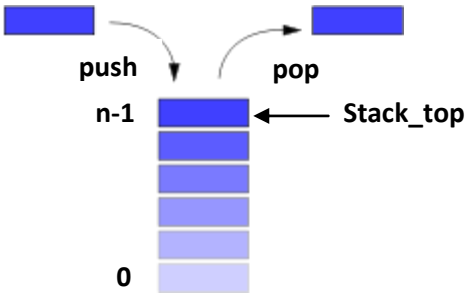
المكدس stack هو تركيبة بيانات خطية linear data structure ، وهو قائمة خطية linear list من العناصر items التي يمكن إضافة insertions وإزالة deletions عنصر من هذه القائمة من نهاية واحدة تسمىTop.

- المكدس stack هو نوع بيانات مجرد abstract data type .
- يتم تطبيق المكدس stack من خلال المصفوفات arrays أو من خلال القوائم المرتبطة linked lists سنتحدث عنها لاحقاً.

االعمليات التي تجرى على المكدس The operations of stack :

المكدس تجرى عليه ثلاث عمليات أساسية three fundamental operations :

1. **عملية الإدخال The 'push' operation :** هذه العملية تقوم بإدخال insert عنصر البيانات data item في قمة المكدس Top.
2. **عملية الإخراج The 'pop' operation :** هذه العملية تزيل delete عنصر البيانات data item من قمة المكدس Top .
3. **عملية The 'peek' operation :** هذه العملية تسمح باختبار عنصر البيانات data item الموجود في قمة المكدس Top بدون إزالته.



لذلك يعتبر المكدس Dynamic list .

المكدس توجد له عدة تطبيقات Stacks have numerous applications :

نحن نرى المكدس stack في حياتنا العادية، على سبيل المثال:

- من الكتب books في مكتبتنا library.
- من حزمة الأوراق sheaf of papers في الطابعة printer.

كل هذه التطبيقات تستخدم في مفهوم أخر واحد يدخل هو أول واحد يخرج (LIFO) .

على سبيل المثال:

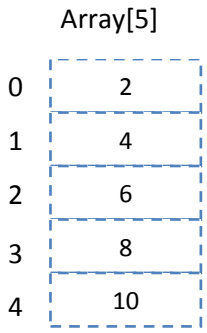
- تخيل أنه لدينا مجموعة من الكتب (كومة من الكتب)، مرصوصة فوق بعضها، أي كتاب و فوقه واحد أخر إلى أن نصل إلى أخر كتاب.
- الآن لكي نضيف كتاب أخر إلى المجموعة، يجب أن نضعه على رأس كومة الكتب، يعني أعلى شيء top .
- وإذا أردنا أن نأخذ أي كتاب يجب أن نسحب الذي فوقه أولاً . أي لا نستطيع سحب الكتاب الرابع مثلاً دون سحب الكتب التي تقع فوقه.

ويتضح من المثال السابق أن المكدس هو عبارة عن فكرة "طريقة" تطبق على المصفوفات ليس في كل الحالات، ولكن سنستخدم المصفوفة هنا، بحيث أن إدخال العناصر يتم من أعلى " كما في حالة الكتب"، وكذلك سحب العناصر يتم من أعلى.

وذلك على خلاف المصفوفة العادية:

مثلاً:

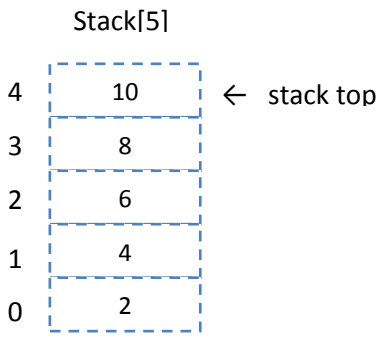
- إذا أدخلنا في أي مصفوفة العناصر ← 2 ، 4 ، 6 ، 8 ، 10



وإذا أردنا عرضها على الشاشة فإن النتيجة هي:

2 ، 4 ، 6 ، 8 ، 10 على نفس ترتيب الإدخال

- ولكن إذا ادخلنا الأعداد السابقة في المكدس stack :



وعرضنا عناصر المكدس stack على الشاشة، فالنتيجة هي:

10 ، 8 ، 6 ، 4 ، 2 أي على عكس ترتيب الإدخال.

- لذلك في المكدس stack أخر واحد يدخل هو أول واحد يخرج (LIFO) .

- LIFO ➔ (Last in, first out)

التطبيق للمكدس Implementation of stack :

في أغلب لغـات البرمجة العاليـة high level languages ، المكدس stack يمكن أن يطبـق بسهولة بواسطـة إما المصفوفة array أو بواسطة القائمة المرتبطة linked list التي سنتحدث عنها لاحقاً.

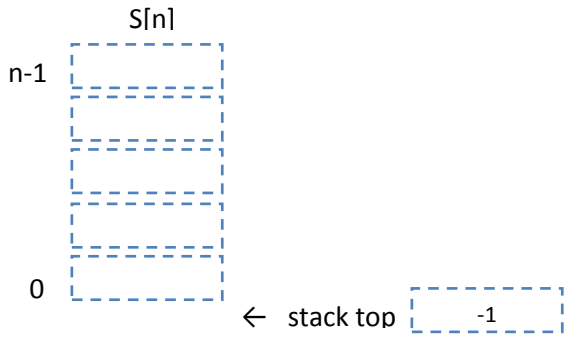
المستخدم يسمح له فقط باستخدام عمليتي 'push'، 'pop' لإدخال و إزالة عنصر item من قمة المكدس stack ، مع بضعة عمليات مساعدة .

س) كيف سنعرف أن العنصر في قمة المكدس ؟

سنحتاج إلى مؤشر للمصفوفة "عدد صحيح int " ، وذلك لكي نعرف من هو أعلى عنصر item ، وليكن اسمه stack_top .

في الحقيقـــة: stack_top ليس مؤشر pointer وإنما هو عدد int ، ولكن نستخدمه كدليل إلى العنصر الأعلى في المصفوفة، يعني إذا كان عندي مصفوفة من 10 عناصر items ، والمستخدم أدخل قيمة أول عنصرين، فإن ال متغير stack_top سيحمل القيمة 1 ، دلالة على أن العنصر الثاني هو أعلى عنصر.

```
CREATE S[n] , stack_top ← -1
```




```
INSERT ( S[n] , stack_top , x )
IF stack_top = n-1 , THEN "Stack Full"
 stack_top ← stack_top + 1
S[stack_top] ← x
END
```

```
DELETE ( S[n] , stack_top )
IF stack_top = -1 , THEN "Stack Empty"
stack_top ← stack_top - 1
END
```

تطبيق المكدس بواسطة المصفوفة Implementing a stack with an array :

دعنا نفكر كيف يتم تطبيق المكدس stack في لغة السي C programming language .

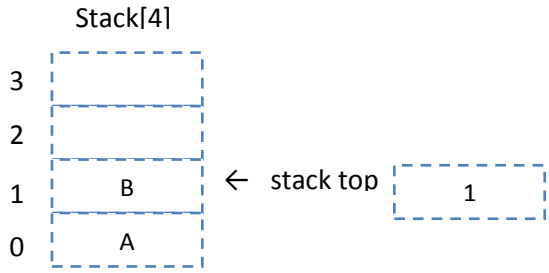
أولاً ، إذا أردنا أن نخزن حروف letters ، نحن يمكن أن نستخدم النوع char ، ولكون المكدس stack يحمل عادةً مجموعة من العناصر items من نفس النوع (على سبيل المثال، char) يمكن أن نستخدم المصفوفة لحفظ محتويات المكدس stack .

نفرض أننا اخترنا مصفوفة حجمها 4 ، هل هذه المصفوفة كافية أم سنحتاج لتخزين المزيد من المعلومات التي تتعلق بتطبيق المكدس stack بواسطة المصفوفة ؟

الإجابة: نحتاج لحفظ المسار لقمة المكدس stack لذلك سنستخدم متغير من النوع الصحيح int وهو stack_top الذي سوف يحمل فهرس المصفوفة index للعنصر الذي يوجد في قمة المكدس stack .

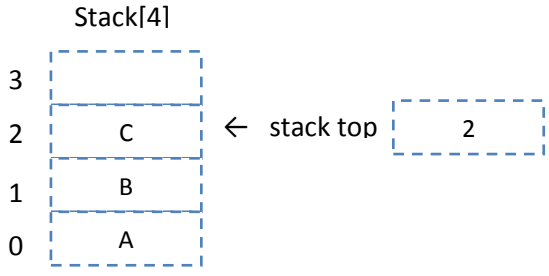
مثال:

- نفرض أن المكدس stack يوجد فيه عنصران (A , B)



لكون B في قمة المكدس stack ، قيمة stack_top تخزن الفهرس index لـ B في المصفوفة (على سبيل المثال: 1).

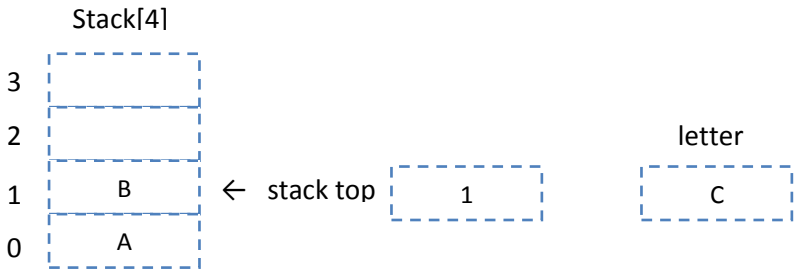
- الآن نفرض بأننا ندخل عنصر في المكدس stack ، push(stack, 'C') سنحصل على:



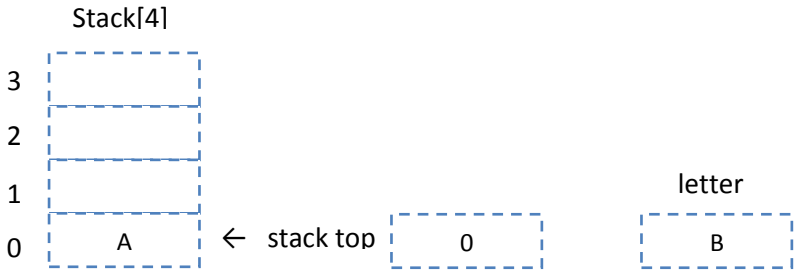
نلاحظ بأن كل من محتوى المكدس stack و قمة المكدس stack_top يتغير .

- نفرض أننا قمنا بسلسلة الإزاحات التالية:

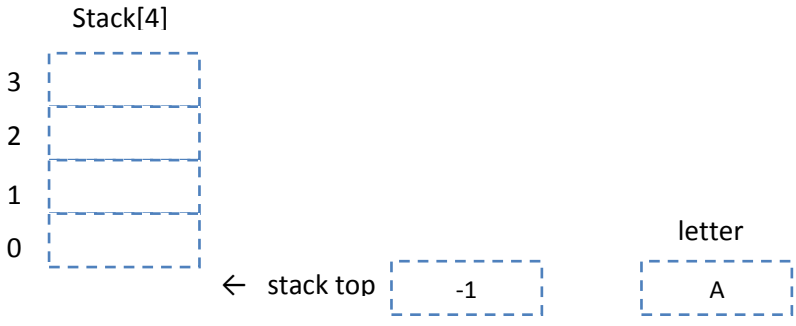
1. Letter = pop (stack)



2. Letter = pop (stack)



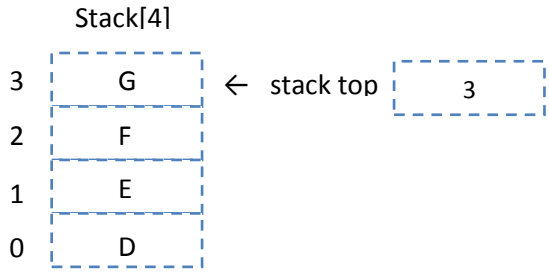
3. Letter = pop (stack)



بذلك أصبح المكدس stack فاضي empty

- ماذا يحدث إذا قمنا بمجموعة العمليات التالية؟

1. push(stack, ‘D’)
2. push(stack, ‘E’)
3. push(stack, ‘F’)
4. push(stack, ‘G’)



عند محاولة إضافة ’H‘ نجد أن المكدس stack أصبح ممتلئ full .

push(stack, ‘H’)

برنامج بلغة السي لتطبيق المكدس بواسطة المصفوفة Code with C language for implementing a stack with an array :

العمليات على المكدس Operations on Stacks :

- **createstack(s)** : لتهيئة s كمكدس فاضي stack empty .
- **push(s,i)** : لدفع/إدخال عنصر i في المكدس stack s .
- **pop(s)** : للوصول وإزالة عنصر في قمة المكدس stack s .
- **peek(s)** : الوصول إلى العنصر الذي في قمة المكدس stack s بدون إزالته.

إعلان المكدس الذي يستخدم المصفوفة Stack Declaration Using an Array :

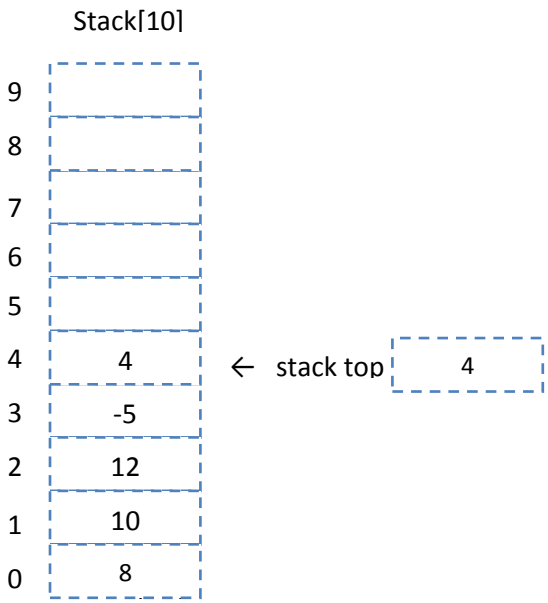
نفرض أن عناصر المكدس stack هي من النوع الصحيح int والمكدس stack يمكن أن يخزن 10 عناصر كحد أقصى.

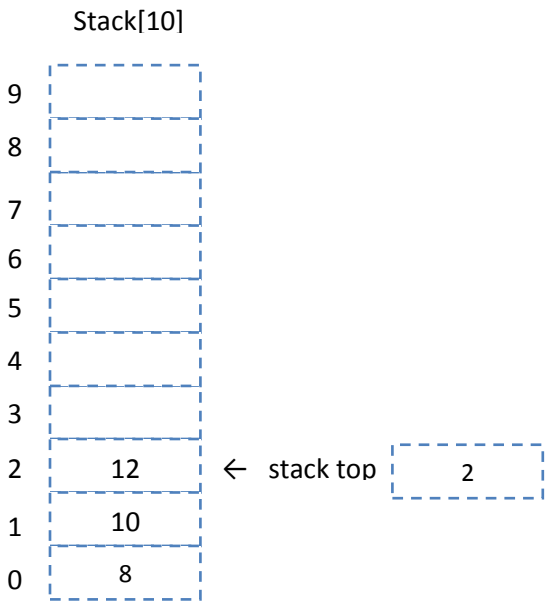
```
#define MAX 10

typedef struct
{
    int top;
    int elements[MAX];
}stack;
stack *ps;
```

- هنا نحن عرفنا نوع بيانات data type خاص يسمى المكدس stack .
- العنصر الأول 'top' يستخدم لفهرسة العنصر الأعلى في المكدس stack .
- المصفوفة 'elements' تحتفظ بعناصر المكدس stack .
- الخط الأخير يعلن مؤشر ps* من نوع المكدس stack .

تمثيل المكدس في الذاكرة Representation of Stack in Memory :





تكوين مكدس فاضي Creating an Empty Stack :

- قبل أن نستخدم المكدس stack ، من الضروري أن يتم تهيئته.
- الدليل/الفهرس للمصفوفة 'elements' يمكن أن تأخذ أي قيمة في المدى من 0 إلى MAX−1 ، الغرض من تهيئة المكدس stack هو تخصيص قيمة 1− للمتغير top الذي يشير إلى العنصر في قمة المكدس stack .
- هذه المهمة البسيطة يمكن أن تنجز بالوظيفة التالية:

```
void createstack( stack *ps)
{
    ps->top = -1;
}
```

عملية الدفع/الإدخال Push Operation :

قبل عملية الدفع /الإدخال push ، إذا المكدس stack فاضي، إذا قيمة المتغير 'top' ستكون الفهرس/الدليل index للعنصر الذي حالياً في المتغير 'top'. لذا عندما نضع القيمة في المكدس stack ، قيمة المتغير 'top' تزيد لكي تشير إلى العنصر الجديد الذي في قمة المكدس stack.

```
void push(stack *ps, int value)
{
    if(ps->top==MAX-1)     // Testing if the Stack is Full اختبار إذا المكدس ممتلئ
       printf("Stack is Full);
    else{
        ps->top++;
        ps->elements[ps->top]=value;
         }
}
```

عملية الإخراج Pop Operation :

العنصر في قمة المكدس stack يخصص لمتغير محلي local variable ، الذي لاحقاً سيعود عن طريق جملة العودة return statement . بعد تخصيص العنصر في قمة المكدس stack لمتغير محلي local variable ، المتغير 'top' تنقص قيمته، لكي يشير إلى عنصر جديد في قمة المكدس stack .

```
void pop(stack *ps)
{
    if(ps->top==-1)     // Testing if the Stack is Empty اختبار إذا المكدس فاضي
       printf("Stack is Empty);
    else
       ps->top--;
}
```

الوصول الى العنصر في قمة المكدس Accessing the Top Element :

هناك قد تكون حالات نريد فيها الوصول إلى العنصر في قمة المكدس stack بدون إزالته من المكدس stack .

```
int peek( stack *ps)
{
    return(ps->elements[ps->top]);
}
```